Synthesis of Petri Nets from Scenarios with VipTool

Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser*

Department of Applied Computer Science,
Catholic University of Eichstätt-Ingolstadt
firstname.lastname@ku-eichstaett.de

Abstract. The aim of this tool paper is twofold: First we show that VipTool [9,2] can now synthesize Petri nets from partially ordered runs. To integrate this extension and further new functionalities, we changed the architecture of VipTool to a flexible plug-in technology. Second we explain how VipTool including the synthesis feature can be used for a stepwise and iterative formalization and validation procedure for business process Petri net models. The synthesis functionalities fill a gap in a previously defined procedure [9,7] where the first definition of an initial process model had to be done "by hand", i.e. without any tool support.

1 Introduction

Automatic generation of a system model from a specification of its behaviour in terms of single scenarios is an important challenge in many application areas. Examples of such specifications occurring in practice are workflow descriptions, software specifications, hardware and controller specifications or event logs recorded by information systems.

In the field of Petri net theory, algorithmic construction of a Petri net model from a behavioural specification is known as synthesis. *Synthesis of Petri nets* has been a successful line of research since the 1990s. There is a rich body of nontrivial theoretical results, and there are important applications in industry, in particular in hardware system design, in control of manufacturing systems and recently also in workflow design.

The classical *synthesis problem* is the problem to decide whether, for a given behavioural specification, there exists an unlabelled Petri net, such that the behaviour of this net coincides with the specified behaviour. In the positive case, a synthesis algorithm constructs a witness net. For practical applications, the main focus is the computation of a system model from a given specification, not the decision of the synthesis problem. Moreover, applications require the computation of a net, whether or not the synthesis problem has a positive answer. To guarantee a reasonable system model also in the negative case, in this paper, the synthesized model is demanded to be the best (a good) upper approximation, i.e. a net including the specified behaviour and having minimal (few) additional behaviour (for details see [12,1]).

Theoretical results on synthesis mainly differ w.r.t. the considered *Petri net class* and w.r.t. the considered model for the *behavioral specification*. Synthesis can be applied to various classes of Petri nets, including elementary nets [10,11], place/transition nets (p/t-nets) [1] and inhibitor nets [13]. The behavioural specification can be given by a

* Supported by the project "SYNOPS" of the German Research Council.

transition system representing the sequential behaviour of a system or by a step transition system additionally considering steps of concurrent events [1]. Synthesis can also be based on a language. Languages can be finite or infinite sets of occurrence sequences, step sequences [1] or partially ordered runs [12]. Despite of this variety, the synthesis methods follow one common theoretical concept, the so called *theory of regions*.

Although there is a wide application field, there is only few tool support for Petri net synthesis so far. Synthesis of labelled elementary nets from transition systems is implemented in the tool Petrify [6]. Petrify is tailored to support hardware system design by synthesizing asynchronous controllers from signal transition graphs (both modelled as Petri nets). The process mining tool ProM [15] uses synthesis methods to automatically generate a process model (given by a Petri net) from so called event logs. Event logs specify a set of occurrence sequences. But ProM does not offer pure Petri net synthesis algorithms. Synet [5] is a software package synthesizing bounded p/t-nets from transition systems. Synet supports methods for the synthesis of so called distributable Petri nets (consisting of distributed components which communicate by asynchronous message passing). It was applied in the area of communication protocol design. However, the authors point out that the tool should be considered as a preliminary prototype rather than a stable and robust tool.

This paper presents a new synthesis tool. The tool supports for the first time synthesis of place/transition-nets from a set of partially ordered runs. The set of partially ordered runs models alternative executions of a place/transition-net. The tool is implemented as an extension of VipTool [9,2], which was originally designed at the University of Karlsruhe within the research project VIP (Verification of Information systems by evaluation of Partially ordered runs) as a tool for modelling, simulation, validation and verification of business processes using (partially ordered runs of) Petri nets. With this new synthesis package all aspects concerned with partially ordered runs of Petri nets – namely synthesis, unfolding (combined with respective validation) and testing of executability (see Figure 1) – are covered by VipTool. So far, the synthesis package is restricted to synthesis from finite sets of partially ordered runs as described in [12]. Typical applications of synthesis only have to deal with finite specifications. Since occurrence sequences and step sequences are special cases of partially ordered runs, the synthesis algorithms are applicable for all kinds of finite languages. Two conceptually different synthesis methods are implemented, covering the major language based synthesis approaches described in the literature [13]. The main approaches to deal with finite representations of infinite languages will be implemented in the near future.

Due to the growth of the number of functionalities of VipTool, we redesigned VipTool in a flexible open plug-in architecture. Algorithms are implemented as plug-ins.

There are many Petri net tools for modelling and analysis, but VipTool occupies a special niche: VipTool is the only tool offering a comprehensive bundle of methods (see Figure 1 for an overview) concerned with causality and concurrency modelled by partially ordered runs of Petri nets (other tools concerned with partial order behaviour of Petri nets focus on model checking techniques employing unfoldings, but not on causality and concurrency). In contrast to occurrence and step sequences, partially ordered runs allow to represent arbitrary concurrency relations between events. Therefore, they are very well suited for the modelling of scenarios of concurrent systems. Advantages

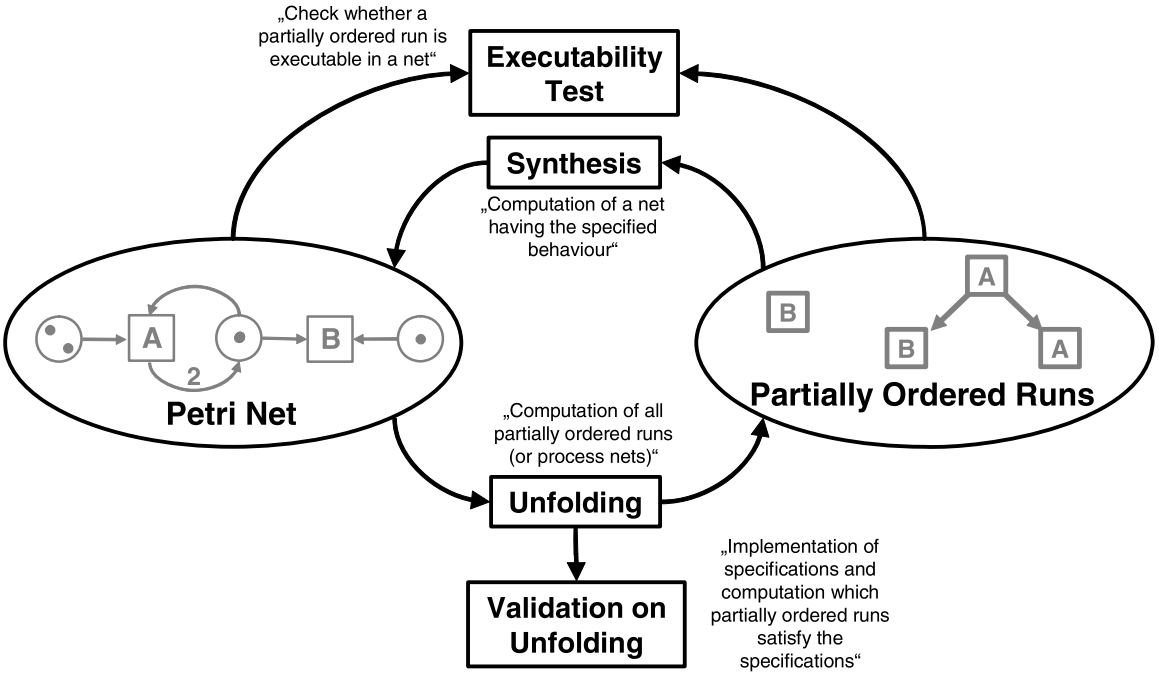

Fig. 1. Sketch of the key functionalities of VipTool

over sequential scenarios are in intuition, efficiency and expressiveness [7,9]. In particular, to specify system behaviour, instead of considering sequential scenarios and trying to detect possible concurrency relations from a set of sequential scenarios, it is much easier and intuitive to work with partially ordered runs.

Starting with a specification of a distributed system in terms of partially ordered runs, the synthesis package of VipTool is used for the generation of prototypes, to uncover system faults or additional requirements using Petri net analysis, to evaluate design alternatives, to visualize a specification or even for the automatic generation of final system models (see [7,8,9,14] for the significance of scenarios in requirements engineering and system design). In the remainder of this section, we explain how to apply the synthesis package in business process design and how the new functionalities work together with the existing validation and verification concepts of VipTool.

One of the main issues of modelling a business process is analysis. Obviously, analysis requires a high degree of validity of the model with respect to the actual business process in the sense that the model faithfully represents the process. For complex business processes, a step-wise design procedure, employing validation of specifications and verification of the model in each step, was suggested in [7,9]. As shown in [2,9], so far VipTool supported most steps of this approach (see also Figure 1). It generates occurrence nets representing partially ordered runs of Petri net models. Specifications can be expressed on the system level by graphical means. Occurrence nets are analyzed w.r.t. these specified properties. Runs that satisfy a behavioural specification are distinguished from runs that do not agree with the specification. The algorithms of VipTool for testing executability of scenarios offer functionalities for detailed validation and verification of the model or a specification w.r.t. single runs. A complicated step that is not supported by previous VipTool versions is the creation of an initial Petri net model for the design procedure. The classical approach to develop a process model is identifying

tasks and resources of the process and then directly designing the control-flow. The validity of the model is afterwards checked by examining its behaviour in comparison to the behaviour of the business process. Using synthesis algorithms (as supported by VipTool) the procedure changes [8]. The desired behaviour of the model constitutes the initial point. First scenarios (in some contexts also called use cases) of the business process are collected. Then the process model is automatically created. In this approach the main task for the user is to design appropriate scenarios of the process (exploiting descriptions of known scenarios that have to be supported by the business process). Usually, it is less complicated to develop and model single scenarios of a process than directly modelling the process as a whole. In particular, in contrast to a complete process model, scenarios need not be designed by some modelling expert, but they may also be designed independently by domain experts.

In Section 2 we survey the new architecture and the features of VipTool. In Section 3 the new synthesis functionalities are illustrated with a small case study.

2 Architecture and Functional Features of VipTool

In this section we provide a detailed description of the functionalities and the architecture of VipTool. VipTool is a platform independent software tool developed in Java. Its previous versions [9,2] were standalone applications providing functionalities for the analysis of partially ordered runs of Petri nets based upon a Java Swing GUI. Due to the growth of functionalities and to increase feature extensibility we redesigned VipTool as an open plug-in architecture. The focus is still on algorithms concerned with partially ordered runs. VipTool uses only standard Java libraries and a GUI developed with the Java swing widget toolkit. It is implemented strictly following advanced object oriented paradigms. In the development of VipTool, we accounted for professional standards such as design patterns, coding conventions, a bug-tracking system and an extensive documentation. The following paragraph gives an overview of the new architecture of VipTool.

The VipTool core platform offers a flexible plug-in technology. The so-called extension manager serves as the foundation for the integration of plug-ins. More precisely, it provides functions to load plug-ins into VipTool. Additionally, the core platform provides definition and implementation of basic GUI elements and related actions, project management functionalities, job scheduling organization as well as system-wide configurations. Development of plug-ins is supported by the VipTool SDK library. VipTool SDK provides appropriate interfaces that can be implemented by plug-ins as well as interfaces arranging the connection to the VipTool core platform. VipTool SDK also includes some key support classes. Embedding of plug-ins into the GUI of the core platform is managed via xml-specifications supporting easy extensibility and scalability. That means, xml-files are used to configure the GUI of VipTool by adequately adding and extending menus, buttons, etc. relating to respective plug-ins. The functional components of VipTool are arranged in an open plug-in architecture connectable to the core platform by the extension manager. Figure 2 depicts this architecture, where the functional component plug-ins are bundled to packages including homogeneous functionalities. Component dependencies are indicated by arrows.

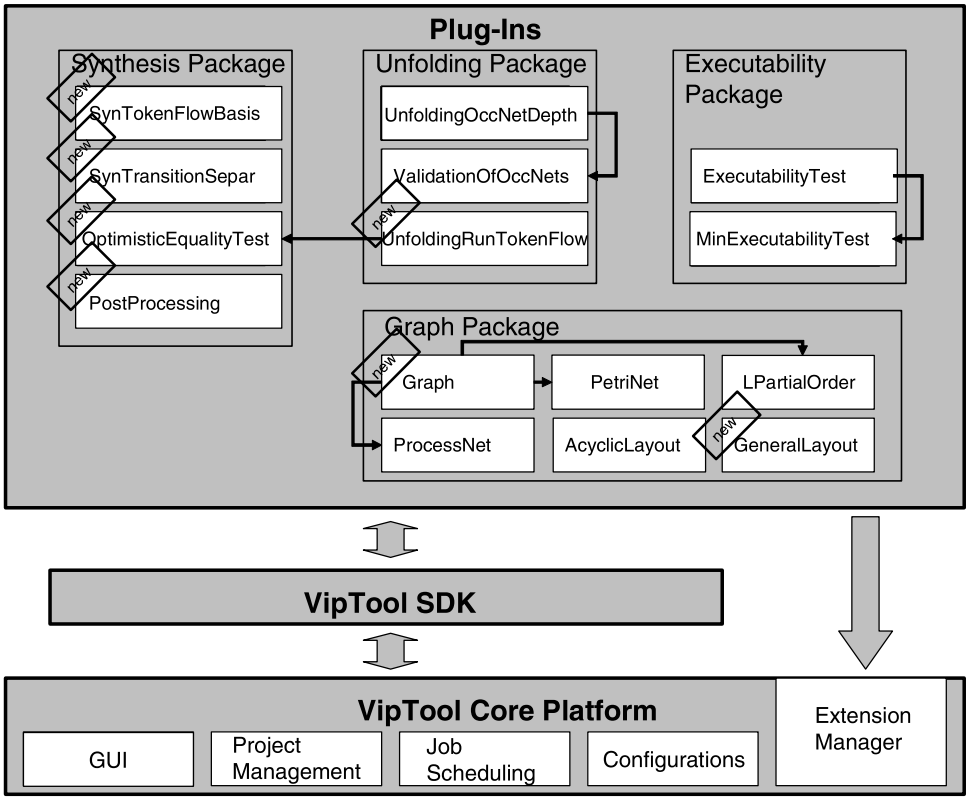

Fig. 2. Architecture of VipTool

The key functionalities of previous VipTool versions have been extracted and reimplemented in independent plug-ins. In Figure 2, new components of VipTool are distinguished from reimplemented functionalities by a ”new”-label. Since the number of supported graph and Petri net classes grew, we decided to define graph classes as general as possible in a special plug-in and to implement editing and user interaction functionalities for certain graph classes as additional plug-ins (”Graph Package”). Plug-ins providing algorithmic functionalities (”Synthesis Package”, ”Unfolding Package”, ”Executability Package”) can be executed in different threads using the job system of the core platform. Plug-ins may communicate with the core platform according to arbitrary communication patterns. The implementation of useful standard communication patterns for algorithmic plug-ins such as status messages, progress bars and logging information is supported by special interfaces of VipTool SDK. Algorithms can manipulate data by using common datastructures defined in Java classes and interfaces of certain plug-ins. To simplify integration of components developed independently or cooperatively by different teams, VipTool also supports easy data exchange between plug-ins and the core platform using xml-files, e.g. pnml-files in the case of Petri nets (pnml is a widely acknowledged standard exchange format for Petri nets) and lpo-files in the case of partially ordered runs (lpo is an xml-file format in the style of pnml). Thus, common datastructures among plug-ins are not required, facilitating extensibility, scalability, composability and reusability of VipTool functionalities. Respective xml-files are arranged by the core platform in workspaces and projects, allowing an arbitrary subfolder structure. The core platform supports a project tree window to offer easy file

management. This is important for flexible and easy invocation of algorithms because the various functionalities have different and partly complex input and output types.

Short descriptions of the VipTool plug-ins shown in Figure 2:

Graph Package

Graph: Provides basic graph classes and interfaces. This plug-in forms the foundation of the ”PetriNet”, ”LPartialOrder” and ”ProcessNet” plug-in.

PetriNet: Includes Petri net visualization and editing functionalities as well as simple interactive features such as playing the token game and showing pre- and post-sets.

LPartialOrder: Supports visualization and design of partially ordered runs by labelled partial orders and offers some related functionalities such as computing the transitive closure of a binary relation or the skeleton of a partial order.

ProcessNet: Provides visualization functionalities for occurrence nets.

Acyclic Layout: Offers automatic layout functionalities for directed acyclic graphs such as labelled partial orders and occurrence nets (based on the Sugiyama algorithm).

General Layout: Offers automatic layout functionalities for general graphs such as Petri nets (based on the spring embedder algorithm by Fruchterman and Reingold).

Synthesis Package

SynTokenFlowBasis: Implements the constructive part of the synthesis algorithm for place/transition Petri nets from a finite set of partially ordered runs as described in [4,12]. So called token flow regions and a finite representation by basis regions are applied [13] (employing the algorithm of Chernikova). The result of the algorithm is a net representing a best upper approximation to the specified behaviour.

SynTransitionSepar: Implements the constructive part of a synthesis algorithm for place/transition Petri nets from a finite set of partially ordered runs using a translation to step sequences (based on methods described in [1]). So called transition regions and a finite representation by separating regions are applied [13] (employing the Simplex method). The result of the algorithm is either a negative answer to the synthesis problem combined with a net, illustrating the reason for the negative answer, or a net representing a best upper approximation to the specified behaviour. In the first case the computed net is a good upper approximation but not necessarily a best upper approximation to the specified behaviour (although a best upper approximation exists).

OptimisticEqualityTest: Implements the optimistic equality test described in [4,12] using the newly developed unfolding plug-in ”UnfoldingRunTokenFlow” (employs a graph isomorphism test between single partially ordered runs by a branch-and-bound procedure optimized for partially ordered runs [4]). It shows if the behaviour of a net synthesized by the ”SynTokenFlowBasis” or the ”SynTransitionSepar” plug-in matches the specified behaviour. In the positive case the synthesis problem has a positive answer, otherwise a negative answer.

PostProcessing: Contains a simple and very fast method to delete implicit places from a net. This reduces the size of nets synthesized with the ”SynTokenFlowBasis” or the ”SynTransitionSepar” plug-in. More advanced post-processing methods are planned.

Unfolding Package
UnfoldingOccNetDepth: Unfolds a Petri net to its occurrence nets (following standard techniques). Occurrence nets are constructed on the fly in a depth first order. Also construction of the branching process including cut-off criteria is supported. See also [9].
ValidationOfOccNets: Allows to specify graphically certain properties of a Petri net, like specific forms of forbidden and desired behaviour. The set of runs, computed by the ”UnfoldingOccNetDepth” plug-in, is divided into these runs, which fulfill the specifications, and runs, which do not fulfill at least one of the specifications. See also [9].
UnfoldingRunTokenFlow: Implements the unfolding algorithm to construct the set of all partially ordered runs of a Petri net described in [3]. The algorithm applies an innovative unfolding concept based on token flows, which in the case of general place/transition-nets is considerably superior to standard unfolding algorithms in time and memory consumption. This is in particular important to improve the runtime of the ”OptimisticEqualityTest”, which depends on an unfolding algorithm. The algorithm does not regard cut-off criteria.

Executability Package
ExecutabilityTest: Supports the polynomial test, whether a given partially ordered run is executable in a given Petri net, explained in [2] (employing methods from the theory of flow networks). The plug-in facilitates failure analysis and constructs an occurrence net corresponding to a checked partially ordered run. See also [2].
MinExecutabilityTest: Offers an algorithm to compute whether a partially ordered run, executable in a Petri net, is minimal executable (based on the plug-in ”ExecutabilityTest”). See also [2].

3 Case Study

We briefly illustrate the new synthesis functionalities of VipTool by a simple case study. We consider the workflow caused by a damage report in an insurance company, i.e. how a claim is processed. The workflow is described by possible (alternative) scenarios of the business process represented by partially ordered runs (note that it is enough to consider maximal runs with minimal causality). A Petri net model of the business process is automatically generated either by the ”SynTokenFlowBasis” plug-in or the ”SynTransitionSepar” plug-in.

Figure 3 shows the partially ordered runs modelled in VipTool and the nets computed with the synthesis plug-ins. There are three possible scenarios: All start with the registration of the loss form submitted by the client (task ”Register”), followed by two concurrent tasks ”Check Damage” and ”Check Insurance”. The latter models checking validity of the clients insurance, the former represents checking of the damage itself. Scenario 1 models the situation that both checks are evaluated positively, meaning that the damage is payed (task ”Pay Damage”) after the two checks. If one evaluation is negative, the company sends a refusal letter. Thus the task ”Send Refusal Letter” is either performed after a negative evaluation of the task ”Check Damage” (scenario 2) or after a negative evaluation of the task ”Check Insurance” (scenario 3).

Combining these three scenarios to a Petri net by synthesis algorithms yields the net damageReportTF in the case of the ”SynTokenFlowBasis” plug-in (and the ”PostPro-

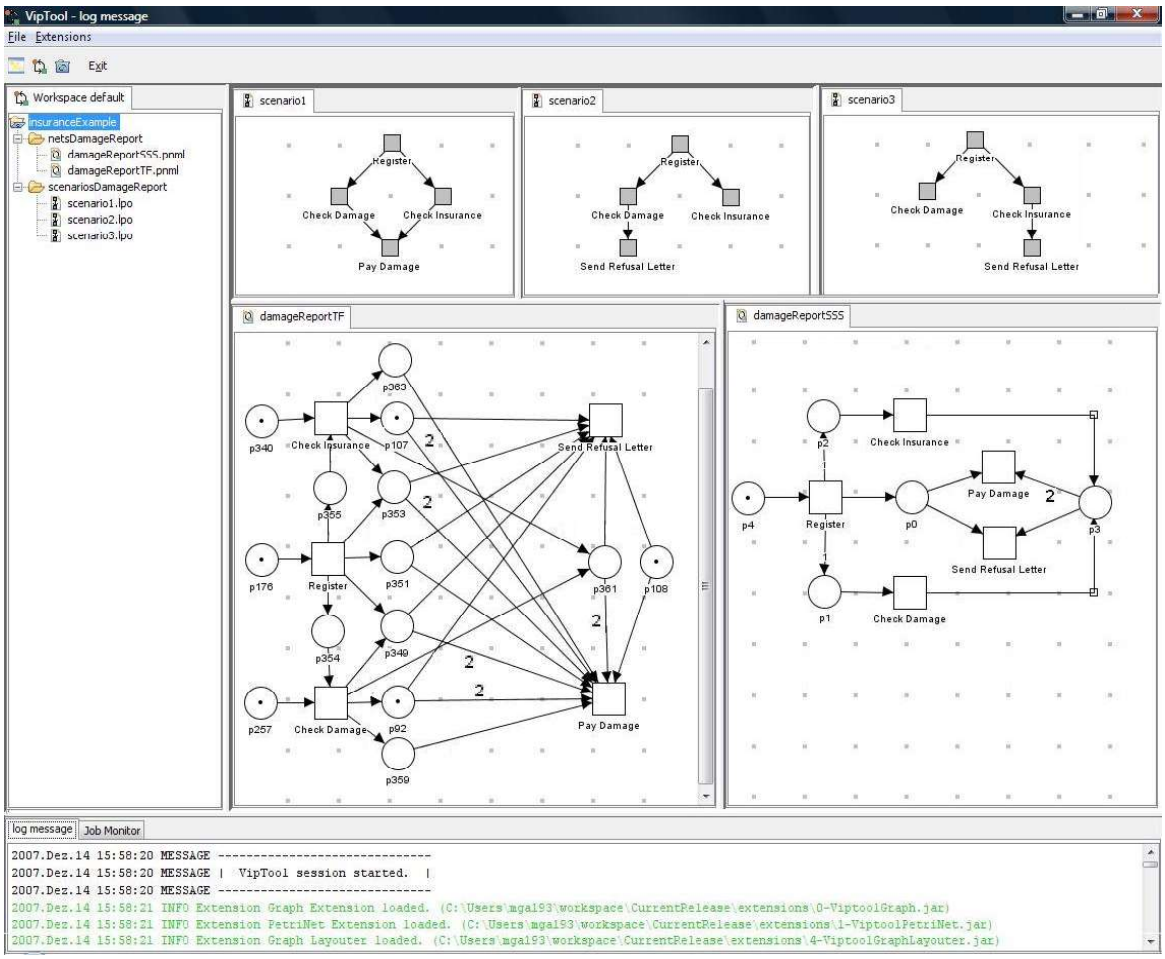

Fig. 3. Screenshot of VipTool showing the standard user interface

cessing" plug-in) and the net damageReportSSS in the case of the "SynTransitionSepar" plug-in. Both nets faithfully represent the considered business process. The test by the "OptimisticEqualityTest" plug-in is positive in both cases. While the net damageReportSSS is nearly the simplest Petri net implementation of the considered business process, the net damageReportTF is complicated. This shows that the "SynTokenFlowBasis" plug-in requires advanced post-processing methods for better readable results.

Figure 4 depicts the scenarios of a more complex variant of the above workflow. The refusal letter can still be sent after the completion of both parallel "Check" tasks if one is evaluated negatively (scenario 1). But if a negative evaluation of one "Check" task already causes sending a refusal letter, while the other "Check" task has not been performed yet, this second "Check" task has to be disabled (i.e. it does not occur in a respective scenario), since it is no longer necessary (scenario 2 and 3). If both "Check" tasks are evaluated positively, an acceptance letter is sent (scenario 4-6). Then either the damage is immediately payed (scenario 4) or additional queries to improve estimation of the loss are asked one (scenario 5) or two (scenario 6) times before the damage is payed. Additionally all six scenarios include the task "Set Aside Reserves". This task is performed after the "Register" task in a subprocess concurrent to all other tasks except for "Pay Damage". Since the damage is payed from the respective reserves, the reserves

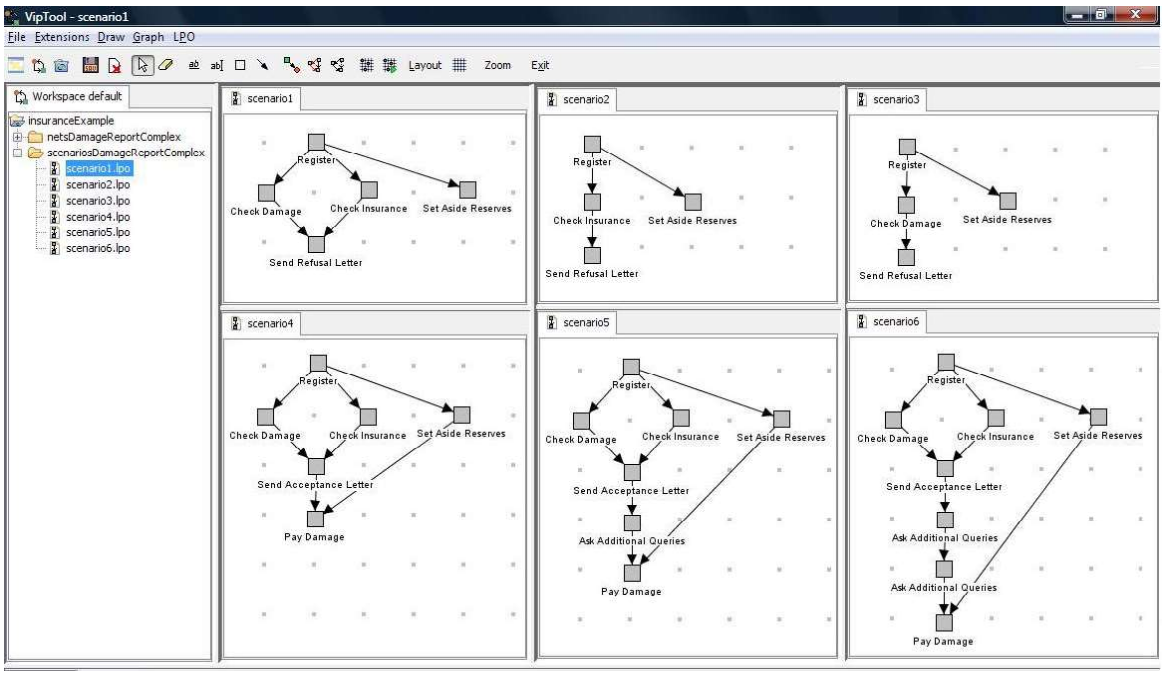

Fig. 4. Screenshot of VipTool showing the user interface of the editor for partially ordered runs

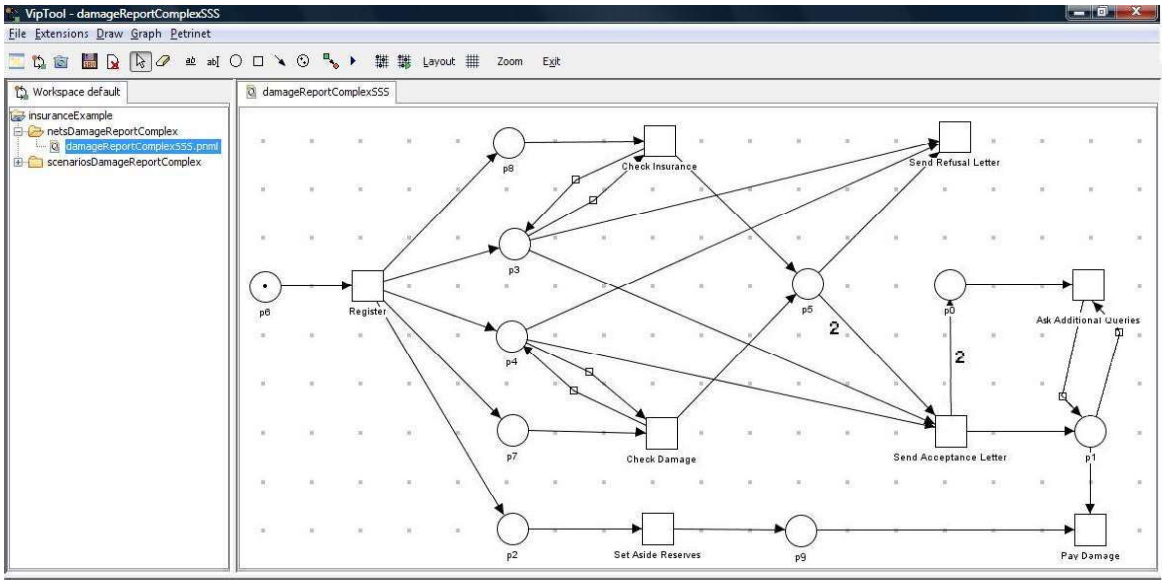

Fig. 5. Screenshot of VipTool showing the user interface of the editor for Petri nets

have to be built up first. Reserves are set aside in any situation, since, in the case the insurance company rejects paying, the reserves have to cover the risk of a lawsuit.

Figure 5 shows the net damageReportComplexSSS synthesized with the ”SynTransitionSepar” plug-in from the set of partially ordered runs depicted in Figure 4. The net represents a very compact model of the described complex business process. The ”OptimisticEqualityTest” yields a positive answer.

The example gives an intuition for our assumption that directly designing a Petri net model of a business process is often challenging, while modelling scenarios and

synthesizing a net is easy. Manually developing a complex Petri net such as the net damageReportComplexSSS for the described business process is an error-prone task, but the design of the six scenarios 1-6 is straightforward, yielding automatically a Petri net by synthesis algorithms.

4 Conclusion

In this paper we surveyed the new synthesis package of VipTool and its applicability in business process design. Discussion of the computational complexity of the implemented synthesis algorithms and experimental results (showing the limitations of the algorithms) can be found in [4] and in a journal version of [12] accepted for Fundamenta Informaticae. The current version of VipTool can freely be downloaded from "http://www.ku-eichstaett.de/Fakultaeten/MGF/Informatik/Projekte/Viptool".

Next steps in the development of VipTool are the implementation of functionalities tuning VipTool to better practical applicability. This includes methods to

- improve the performance of the algorithms of the "Unfolding", "Synthesis" and "Executability" package,
- improve the "PostProcessing" plug-in,
- include further synthesis variants,
- further improve editing functionalities of the "Graph" package,
- deal with high-level nets.

We acknowledge the work of all other members of the VipTool development team: Thomas Irgang, Leopold von Klenze, Andreas Klett, Christian Kölbl, Robin Löscher.

References

1. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
2. Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? Viptool Tells You! In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 381–390. Springer, Heidelberg (2006)
3. Bergenthum, R., Lorenz, R., Mauser, S.: Faster Unfolding of General Petri Nets. In: AWPN 2007, pp. 63–68 (2007)
4. Bergenthum, R., Mauser, S.: Experimental Results on the Synthesis of Petri Nets from Partial Languages. In: Petri Net Newsletter, vol. 73, pp. 3–10 (2007)
5. Caillaud, B.: Synet, `http://www.irisa.fr/s4/tools/synet/`
6. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers. IEICE Trans. of Informations and Systems E80-D(3), 315–325 (1997)
7. Desel, J.: Model Validation - A Theoretical Issue? In: Esparza, J., Lakos, C.A. (eds.) ICATPN 2002. LNCS, vol. 2360, pp. 23–43. Springer, Heidelberg (2002)
8. Desel, J.: From Human Knowledge to Process Models. In: Kaschek, R., et al. (eds.) UNISCON 2008. LNBIP, vol. 5, Springer, Heidelberg (2008)
9. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with Viptool. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 380–389. Springer, Heidelberg (2003)

10. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. Acta Inf. 33(4), 297–315 (1996)
11. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-Structures: Part I + II. Acta Inf. 27(4), 315–368 (1989)
12. Lorenz, R., Bergenthum, R., Desel, J., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. In: ACSD 2007, pp. 157–166. IEEE Computer Society, Los Alamitos (2007)
13. Lorenz, R., Juhás, G., Mauser, S.: How to Synthesize Nets from Languages - a Survey. In: Proceedings of the Wintersimulation Conference (WSC) (2007)
14. Seybold, C., Meier, S., Glinz, M.: Scenario-Driven Modeling and Validation of Requirements Models. In: SCESM 2006, pp. 83–89. ACM, New York (2006)
15. van der Aalst, W.M.P., et al.: ProM 4.0: Comprehensive Support for *real* Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)